"Express Mail" Mailing Label No. **EL960828244US**

PATENT APPLICATION
ATTORNEY DOCKET NO. SUN-P9642-SPL

METHOD AND APPARTUS FOR IMPLEMENTING A LOCK-FREE SKIP LIST THAT SUPPORTS CONCURRENT ACCESSES

Inventor: Paul A. Martin

Related Application

[0001] This application hereby claims priority under 35 U.S.C. §119 to U.S. Provisional Patent Application No. 60/456,792, filed on 21 March 2003, entitled "Practical Lock-Free Skip List," by inventor Paul A. Martin (Attorney Docket No. SUN-P9642PSP).

BACKGROUND

Field of the Invention

[001] The present invention relates to the design of lookup structures within computer systems. More specifically, the present invention relates to a

method and apparatus for implementing a lock-free skip list that supports concurrent accesses within a computer system.

Related Art

[0002] A skip list is a dynamically sized sorted linked list that offers logarithmic time performance for searching, inserting, and deleting elements. William Pugh developed a basic design for a skip list to be used by a single thread (see "A Skip List Cookbook," by William Pugh, University of Maryland Institute for Advanced Computer Studies UMIACS Technical Report No. UMIACS-TR-89-72.1).

[0003] A skip list is neither a single list (which takes linear search time), nor a tree (which requires re-balancing whenever it grows lop-sided in order to avoid requiring linear search time), but instead has an indexing scheme that is incorporated into its basic list structure.

[0004] Nodes in a skip list are mostly just normal linked-list nodes, but a procedure using random numbers chooses to make some of the new additions taller than the baseline--these serve to index into the list with search time proportional to the logarithm of the size of the list. Adjusting the distribution of random numbers selects the base of the logarithm--using more "tall" nodes reduces search time at the expense of the extra space; using shorter average heights is more compact but yields slower searches.

[0005] The head of the skip list is a special sentinel node representing negative infinity and carrying a set of pointers that are as "high" as the tallest node can be. The tail of the skip list is logically a terminator sentinel of similar height representing positive infinity, though it may be simulated by null pointers. The head pointers and all other pointers in the skip list point "forward" to the next node that reaches the "height" of that pointer. The top pointers may be missing,

but once there is a node as tall as a given height, all those pointers "shorter" than it will be filled in.

[0006] Locating a given node (or where a future one will be inserted) is done by following the tallest pointer chain keeping track of a predecessor and successor nodes until the successor node has a value higher than the target node. This process loops by descending to the next lower layer and following the predecessor links at that level until the bottom layer is reached.

[0007] Subsequent to his description of the canonical skip list, Pugh also developed a technique based on locks to use skip lists in a multi-threaded environment (see "Concurrent Maintenance of Skip Lists," by William Pugh, University of Maryland Technical Report No. CS-TR-2222.1 1989). However, when large numbers of processes access a skip list concurrently, contention for locks can become a serious impediment to system performance.

[0008] Hence, what is needed is a method and an apparatus for accessing a skip list in a multi-threaded environment without the performance problems associated with locks.

SUMMARY

[0009] One embodiment of the present invention provides a system that supports concurrent accesses to a skip list that is lock-free. The term "lock-free" means that the skip list can be simultaneously accessed by multiple processes without requiring the processes to perform locking operations (non-blocking), and furthermore that a finite number of operations by a thread will guarantee progress by at least one thread (lock-free). During a node deletion operation, the system receives reference to a target node to be deleted from the skip list. The system marks a next pointer in the target node to indicate that the target node is deleted, wherein next pointer contains the address of an immediately following node in the

skip list. This marking operation does not destroy the address of the immediately following node, and furthermore, the marking operation is performed atomically and thereby without interference from other processes. The system then atomically modifies the next pointer of an immediately preceding node in the skip list to point to an immediately following node in the skip list, instead of pointing to the target node, thereby splicing the target node out of the skip list.

[0010] In a variation on this embodiment, after the target node is spliced out of the skip list, the system modifies the next pointer of the target node so that the next pointer remains marked but points to the immediately preceding node instead of the immediately following node in the skip list.

[0011] In a variation on this embodiment, a node in the skip list can possibly be a tall node that includes one or more higher-level next pointers, wherein a given higher-level next pointer contains the address of the immediately following node in the skip list that reaches or exceeds the height of the given higher-level next pointer. In this variation, if the target node is a tall node, the node deletion operation additionally marks and splices around higher-level next pointers in the target node.

[0012] In a variation on this embodiment, marking the next pointer to indicate that the target node is deleted involves setting a "deleted bit" in the next pointer.

[0013] In a variation on this embodiment, marking the next pointer to indicate that the target node is deleted involves: creating a special node with a deleted type, which points to the immediately following node in the skip list; and atomically replacing the next pointer with a pointer to special node.

[0014] In a variation on this embodiment, during a node insertion operation, which inserts a new node into the skip list, the system locates a node immediately preceding the new node in the skip list. This involves maintaining a

predecessor array, wherein for each level of the skip list, the predecessor array contains a pointer to the node immediately preceding the new node in the skip list. The system also locates a node immediately following the new node in the skip list. This involves maintaining a successor array, wherein for each level of the skip list, the successor array contains a pointer to the immediately following node. Finally, the system splices the new node into the skip list by: setting the next pointer for the new node to point to the immediately following node; atomically updating the next pointer of the immediately preceding node to point to the new node; and if the new node is a tall node, similarly splicing higher-level next pointers associated with the new node.

[0015] In a variation on this embodiment, the system is configured to remove a highest priority node from the skip list through a constant time operation, wherein the head node of the skip list points to the highest priority node for ease of deletion, and wherein keys for nodes are chosen to achieve this ordering.

[0016] In a variation on this embodiment, the system periodically performs a garbage-collection operation to reclaim deleted nodes that have become unreachable.

[0017] In a variation on this embodiment, the target node includes: a key that contains a priority value for the node in the skip list; a value field that contains or points to data associated with the node; a next pointer that contains the address of an immediately following node in the skip list; and zero or more higher-level next pointers, wherein a given higher-level next pointer contains the address of the immediately following node in the skip list that reaches or exceeds the height of the given next pointer.

BRIEF DESCRIPTION OF THE FIGURES

[0018] FIG. 1 illustrates a live node as it might appear in a skip list in accordance with an embodiment of the present invention.

[0019] FIG. 2 illustrates an empty skip list in accordance with an embodiment of the present invention.

[0020] FIG. 3 illustrates an exemplary skip list populated with five visible nodes in accordance with an embodiment of the present invention.

[0021] FIG. 4A illustrates the node deletion process in accordance with an embodiment of the present invention.

[0022] FIG. 4B illustrates an alternative marking process that replaces the next pointer with a pointer to a special delete type node in accordance with an embodiment of the present invention.

[0023] FIG. 5 illustrates the life cycle of a next pointer for a node in a skip list in accordance with an embodiment of the present invention.

[0024] FIG. 6A illustrates operations that take place during the node insertion process in accordance with an embodiment of the present invention.

[0025] FIG. 6B illustrates operations involving higher-level next pointers for the node insertion process in accordance with an embodiment of the present invention.

DETAILED DESCRIPTION

[0026] The following description is presented to enable any person skilled in the art to make and use the invention, and is provided in the context of a particular application and its requirements. Various modifications to the disclosed embodiments will be readily apparent to those skilled in the art, and the general principles defined herein may be applied to other embodiments and applications without departing from the spirit and scope of the present invention. Thus, the

present invention is not limited to the embodiments shown, but is to be accorded the widest scope consistent with the principles and features disclosed herein.

[0027] The data structures and code described in this detailed description are typically stored on a computer-readable storage medium, which may be any device or medium that can store code and/or data for use by a computer system. This includes, but is not limited to, magnetic and optical storage devices such as disk drives, magnetic tape, CDs (compact discs) and DVDs (digital versatile discs or digital video discs), and computer instruction signals embodied in a transmission medium (with or without a carrier wave upon which the signals are modulated). For example, the transmission medium may include a communications network, such as the Internet.

Skip List Node

[0028] FIG. 1 illustrates a live node 100 as it might appear in a skip list in accordance with an embodiment of the present invention. Node 100 includes a key 104, which contains a priority value used to index node 100 within the skip list. Node 100 also includes a value 106, which either contains or points to data associated with node 100. Node 100 additionally includes an array of next pointers 102. The lowest-level next pointer in this array contains the address of an immediately following node in the skip list, whereas higher-level next pointers contain addresses for immediately following nodes in the skip list that reach the height of each higher-level next pointer.

[0029] As was described above, in one embodiment of the present invention, the height of the array of next pointers 102 is determined randomly during the node creation process. Furthermore, every next pointer also includes a "deleted bit," which indicates whether or not node 100 has been deleted as is explained below with reference to the node deletion process. Note that a node is

"deleted" when the bottom (lowest-level) pointer is so marked; deletions of higher-level pointers effectively "shrink" the height of the node.

Empty Skip List

[0030] FIG. 2 illustrates an empty skip list 200 in accordance with an embodiment of the present invention. This empty skip list 200 includes a head node 202 and a tail node 204. Head node 202 and tail node 204 are dummy nodes, which do not contain real data values. The key field of the head node 202 is set to $-\infty$ so that it is smaller than any key in the skip list, and the key field of the tail node 204 is set to $+\infty$ so that it is larger than any key in the skip list. The value fields in the head node 202 and the tail node 204 are set to zero. Note that all values and keys in dummy nodes may be ignored if the implementation checks for equality to the dummy node.

[0031] Head node 202 includes an array of next pointers. This array is configured to be the maximum size of any array of pointers in any node in the skip list. Since the skip list in FIG. 2 is empty, these next pointers are initialized to point to the tail node 204.

Populated Skip List

[0032] FIG. 3 illustrates an exemplary skip list 300 populated with five visible nodes 302-307 in accordance with an embodiment of the present invention. As with empty skip list 200 described above, populated skip list 300 includes both a head node 310 and a tail node 311. Between head node 310 and tail node 311 are a number of visible nodes 301-305. These visible nodes 301-305 contain keys K_1-K_5 and values V_1-V_5, respectively. They also contain next pointer arrays of varying sizes. Note that each next pointer points to the immediately following node that reaches or exceeds the height of the next pointer.

Node Deletion

[0033] FIG. 4A illustrates the node deletion process in accordance with an embodiment of the present invention. FIG. 4A illustrates a target node 404 to be deleted as well as an immediately preceding predecessor node 402 and an immediately following successor node 406.

[0034] This embodiment of the present invention uses a three-stage deletion process. In the first stage, the next pointer of the target node is first marked "deleted" by flipping a special deleted bit in the next pointer. Note that this marking operation involves using an atomic operation, such as a compare-and-swap operation, so that multiple threads cannot interfere with the marking process. Once target node 404 is so marked, all other threads in the system recognize its new status as a deleted node. This allows the other processes to complete the deletion operation if the original process performing the deletion operation stalls.

[0035] In the second stage, the next pointer of predecessor node 402 is changed to point to successor node 406. This operation effectively splices target node 404 out of the linked list.

[0036] In the third stage, the marked-deleted next pointer of target node 404 is updated to remain marked deleted but to point to predecessor node 402 instead of successor node 406. This backwards-pointing next pointer allows a process that has been "stranded in the middle" of a partial deletion to pick up again at the node it would have been operating on if the deletion operation had completed instantly. This optimization can greatly improve the efficiency of the node deletion process. Note that this third stage is not essential. Hence, in one embodiment of the present invention, if the thread that performed the second stage stalls, the third step will not be accomplished until the thread wakes up.

[0037] In one embodiment of the present invention, instead of setting the delete bit to mark the next pointer, the system creates a special "delete type" node 405, which points to the immediately following node in the skip list, and replaces the next pointer in target node 404 with a pointer to special node 405 as is illustrated in FIG. 4B. It is then possible to determine if a node is deleted by looking to see if its next pointer points to a "deleted type" node. In this embodiment, swinging the next pointer of target node 404 back to point to predecessor node 402 involves setting the pointer within special node 405 to point back to predecessor node 402. Note that this technique does not require the garbage collection process to be modified to ignore delete bits.

[0038] A node in the skip list can possibly be a tall node that includes one or more higher-level next pointers, wherein a higher-level next pointer contains the address of the immediately following node in the skip list that reaches or exceeds the height of the higher-level next pointer. If the target node is a tall node, the node deletion operation additionally involves marking and splicing around higher-level next pointers in the target node. With a taller node, the process marks as deleted, splices around, then optionally reverses the highest pointer first. Each lower-level pointer is "deleted" in turn. Deleting the bottom (lowest-level) pointer is the action that really deletes the node; all higher-level deletions merely shorten the node.

[0039] FIG. 5 illustrates the life cycle of a lowest-level next pointer for a node in a skip list in accordance with an embodiment of the present invention. When the node is newly created, the lowest-level next pointer is initially set to a null value (see 502). Next, when the node is linked into the skip list, the lowest-level next pointer is set to point to the immediately following node in the skip list (see 504). Next, the node is deleted by setting the deleted bit in the lowest-level next pointer, and the next pointer remains forward-pointing (see 506). Finally, the

deleted bit remains set while the next pointer is reset to point backwards to the immediately preceding node (see 508). Note that if one thread is "tearing down" a tall node that has not been completely inserted, a "high" pointer may go directly from null to deleted.

Node Insertion

[0040] FIG. 6A illustrates operations involving the next pointer during the node insertion process in accordance with an embodiment of the present invention. As is illustrated in FIG. 6, the node insertion process operates on the skip list, including nodes 601-605. It makes use of a predecessor array 610 to keep track of the immediately preceding node for each level of the skip list, and a successor array 611 to keep track of the immediately following node for each level of the skip list.

[0041] The node insertion process first scans through the skip list to locate predecessor and successor nodes for the new node 606 in the skip list. This can be accomplished by following the tallest pointer chain keeping track of a predecessor and successor node until the successor node has a value higher than the target. This process loops by descending to the next lower layer and following the predecessor links at that level until the bottom layer is reached.

[0042] At the end of this process, the system has populated predecessor array 610 and a populated successor array 611 as is illustrated in FIG. 6A. More specifically, at the first level, predecessor array 610 and successor array 611 indicate that the immediately preceding node is 603 and the immediately following node is 604. Similarly, at the second level, the immediately preceding node is 602 and the immediately following node is 604, and at the third level, the immediately preceding node is 601 and the immediately following node is 605.

[0043] Next, the system splices new node 606 into the skip list by first setting the next pointer for new node 606 to point to the immediately following node 604 (step 1), and then atomically updating the next pointer of the immediately preceding node 603 to point to new node 606 (step 2). If the predecessor pointer has been changed by another thread, this atomic update will fail, and the predecessor array 610 and successor array 611 must be (at least partially) recomputed before retrying the update.

[0044] FIG. 6B illustrates operations involving higher-level next pointers for the node insertion process in accordance with an embodiment of the present invention. If new node 606 is a tall node that includes higher-level next pointers, the system similarly splices the higher-level next pointers associated with new node 606. The process works from the lower levels towards the highest, recomputes predecessor array 610 and successor array 611 if the atomic update fails, and stops if a level has been marked deleted.

Incremental Structure Modification

[0045] In one embodiment of the present invention, the higher parts of a deleted node can be torn down piecemeal by each process that encounters one, recognizing the node's deleted status from the bit in their lowest-level pointer and using the same three-stage pointer deletion technique.

[0046] In this embodiment, the higher parts of a new insertion are built up from the bottom by the thread that inserted it. If this thread stalls, then the higher parts are just not in use--the remainder is a valid node. The cost of this approach is that a number of nodes proportional to the number of dead or stalled insertion processes will exist at any time.

Process Interference

[0047] Note that any number of threads seeking nodes, but not adding or deleting them, can cooperate without interference. Moreover, two or more threads attempting to delete the same node eventually end in a race to be the first to set its deleted flag bit; the first to succeed blocks all the others.

[0048] Any attempt to add new successor to a node that is being deleted is thwarted by the deletion bit on its "next" pointer. If the deletion has set that bit, no successor can be added, and if the successor has been added, the attempt to delete it will be retried.

[0049] In general, the discovery that a node being used at any level for navigation (including the base level) has been deleted calls for a recovery move, and this is accomplished by two mechanisms.

Recovery from Deletions

[0050] The basic recovery technique, when the node being worked on is discovered to have been deleted, is to test whether its "next" pointer has been turned back to point upstream. If so, then following the next pointer will lead to the node that was its predecessor node when it was deleted. If this node is still alive, then the recovery is complete; the predecessor node will point beyond the deleted node.

[0051] Other the other hand, if the node is marked deleted but its "next" pointer still points "forward," then the node that was previously its predecessor may still point to it. If so, the current process takes on the task of splicing around the deleted node and turning the next pointer of the deleted node back.

[0052] If the predecessor found during the search no longer points to the deleted node, then the system retreats to the next higher pointer level and recommences the search from this next-higher level. This "higher ground"

recovery technique is applied recursively until the top level is reached. At this point, a restart from the head sentinel can be used.

[0053] Note that turning back of pointers makes the recovery immediate in most cases and only slightly complicates the design. Moreover, the updating of pointers to turn them around can be accomplished through normal write operations, thereby saving the execution cost of a compare-and-swap instruction.

[0054] One of the uses for a skip list is as a queue that holds an arbitrary number of items at an arbitrary number of priorities. If the skip list is to be used in this manner, the representation of priority or the orientation of the key ordering in the list should be arranged so that the node adjacent to the head dummy is always the one that is next to be popped off, since in this special case the time required for removal of a node is linear rather than logarithmic.

Garbage Collection

[0055] This skip list is described for use in a garbage-collected environment, but the garbage collector needs to be smart enough to recognize that a pointer to X still points to X whether or not an embedded "deleted" bit is set. Alternatively, this list can be run with collection done by the ROP Pass the Buck scheme, but that scheme will need to be extended to guard the nodes pointed to by the head sentinel of the list. (For a description of this ROP scheme see "The Repeat Offender Problem: A Mechanism for Supporting Dynamic-Sized, Lock-Free Data Structures," by Maurice Herlihy, Victor Luchangco and Mark Moir, *Proceedings of Distributed Computing, 16th International Conference, DISC 2002*, Toulouse, France, October 28-30, 2002, pp. 339-353.) Note that if backward pointers are used, an additional extension to track their effects is needed.

[0056] The foregoing descriptions of embodiments of the present invention have been presented only for purposes of illustration and description. They are not intended to be exhaustive or to limit the present invention to the forms disclosed. Accordingly, many modifications and variations will be apparent to practitioners skilled in the art. Additionally, the above disclosure is not intended to limit the present invention. The scope of the present invention is defined by the appended claims.